PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants:	James J. Crow; Dennis L. Parker		
Assignee:	Motive, Inc.		
Title:	BROADBAND SERVICE CONTROL NETWORK		
Application No.:	09/542,602	Filed:	April 4, 2000
Examiner:	Adnan M. Mirza	Group Art Unit:	2141
Docket No.:	MTV0014US	Confirmation No.:	5339

Austin, Texas
April 29, 2005

MAIL STOP AF
COMMISSIONER FOR PATENTS
P. O. Box 1450
Alexandria, VA 22313-1450

DECLARATION OF PRIOR INVENTION IN THE UNITED STATES PURSUANT TO 37 C.F.R. § 1.131

Dear Sir:

PURPOSE OF THE DECLARATION

Claims 16-40 are rejected under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 6,477,580 issued to Bowman-Amuah (Bowman) in view of Johnson et al. (Johnson), U.S. 2002/0095400. The effective date of Bowman is August 31, 1999.

DECLARATION

I, James J. Crow, declare as follows:

1. I am an inventor of the claimed subject matter of the above-referenced application.

2. Attached is Exhibit A. Exhibit A includes a specification for the extensible service bus that we invented. The date on the document of Exhibit A has been removed. The removed date is, however, prior to August 31, 1999.

3. Exhibit A shows a method for managing a plurality of services located on a plurality of servers as an extensible service bus, comprising: providing a registration service where an agent machine can register as a subscriber with the extensible service bus and receive a subscriber identification; providing a login service where the agent machine can connect to the extensible service bus using the subscriber identification; providing a service map management service that receives server location information from each of the plurality of services and generates a service location map comprising a listing of at least one of the plurality of the services included on the extensible service bus and server location information corresponding to each service of the listing; providing a connection status service to monitor the connection status of subscribers and the servers connected to the extensible service bus; and providing a network control service, wherein the network control service causes a setting on a network device to change to establish a network physical connection to the agent machine, and the network physical connection complies with a requirement for the agent machine to use one of the plurality of services.

4. Exhibit A establishes our invention of the subject matter set forth in independent claims 16, 22, 23, 24, and 25 prior to the effective date of Bowman.

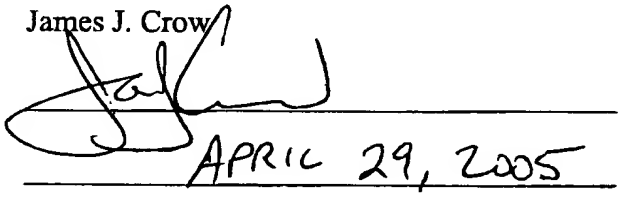
5. The inventive concepts shown in Exhibit A were conceived by Dennis L. Parker and me.

6. All of the enumerated acts took place in the United States of America.

****

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements are made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under §1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the Application or any patent issued thereon.

Full Name of Inventor: James J. Crow

Inventor's Signature: 

Date: APRIL 29, 2005

Country of Citizenship: U.S.A.

Residence: 11301 Monet Drive
Austin, Texas 78726

EXHIBIT A

BroadJump, Inc.

Arch-s-100: Extensible Service Bus specification (speclet)

By Dennis Parker

First draft, ***REDACTED***

Last update ***REDACTED***

Arch-s-100, Extensible Service Bus Specification (speclet)

1 Introduction

This document describes the architecture of the BroadJump product variously identified as "the base server" and the "release 1 server". The features that this product provides are detailed in BroadJump engineering document Feature-s-100. Although it is usually convenient to speak of the product as a server, it is really a set of services provided by a number of distributed server processes.

The features provided to our customers by this product must support a user base that is likely to grow to more than 250,000 active users per installation during the useful life of the product. This level of service can only be provided by a set of services that exhibit a combination of extensibility and scalability to a nearly unprecedented extent. This document describes the key concepts of the architecture that delivers this novel capability.

In this architecture, services are provided in a framework of standardized interconnection mechanisms, reminiscent of the way in which a hardware interface bus works. This makes it possible for a client or service to locate a service using a simple mapping scheme. More complex mapping such as directory structures may be implemented by the services, but these are avoided at the bus level so as to provide the maximum potential flexibility in deployment architectures. In this way it is possible for the deployment architecture to change without interrupting service.

This bus supports the dynamic addition of new services that meet the bus interface rules. In this way a client process may appear as a service on the bus. In order to avoid unmanageable growth in complexity as new services are added, the bus supports partitioning below the top level.

These features of the product lead to its name, the BroadJump Extensible Service Bus or ESB. The base version of ESB includes a number of standard services which implement features of the ESB itself. These are also described in this document.

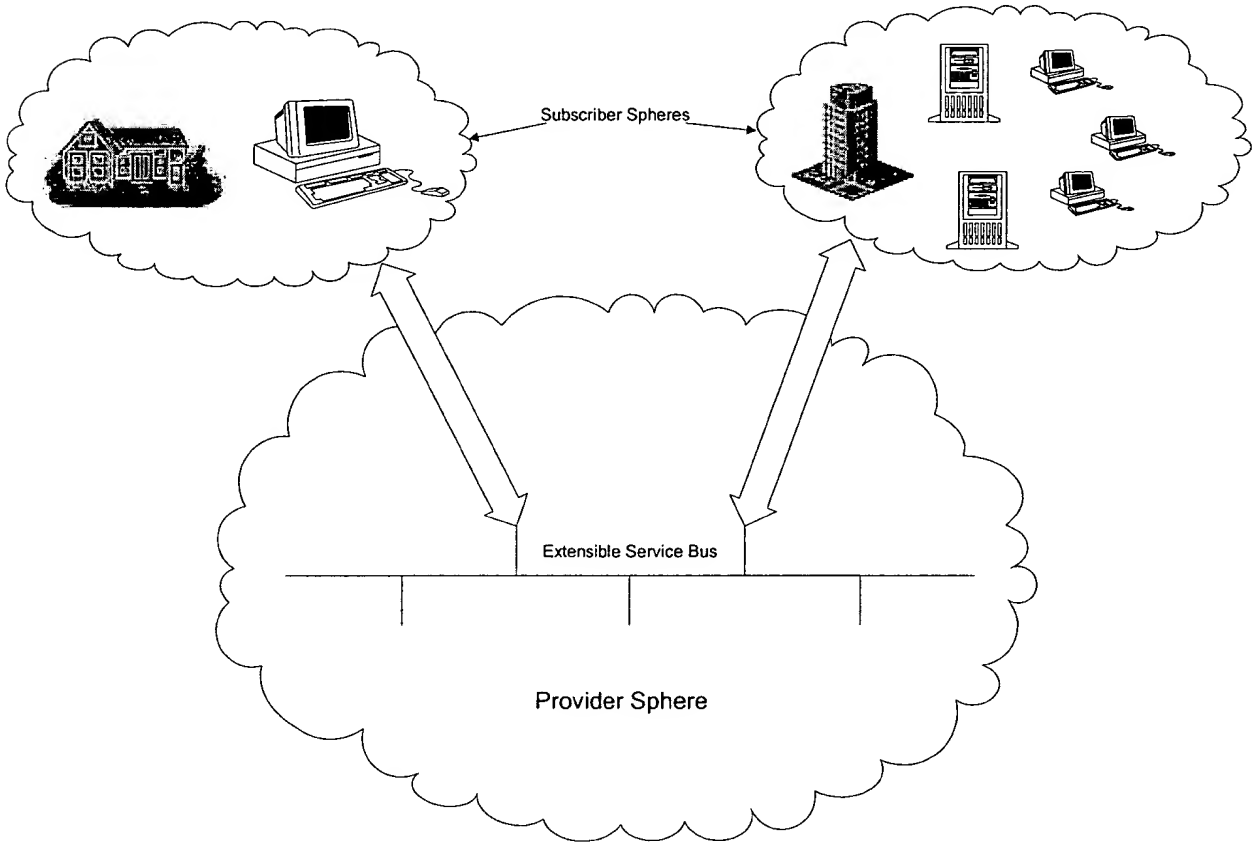
2 The service environment

The typical BroadJump ESB user is an Internet Service Provider, especially using broadband subscriber connections. Most of the subscribers are likely to be individuals, but many will be connecting internal LANs to the Internet via the connection to the Provider. In the later case it may be desirable to host some portion of a service on the subscriber's LAN. Some of the services that the Provider wants to offer to the subscriber may involve direct connections between services that are not part of the initial ESB release.

The ESB architecture allows services to be implemented in processes that run on the Provider's machines, or on subscriber machines. In order to clarify the relationships between these operating environments, it may be helpful to think of each as a self-contained functional unit, a "sphere". There is the Provider Sphere, which includes all the hardware at the Provider's sites, and the various Subscriber Spheres, which include the hardware at the subscriber sites. Additional spheres that might be connected include the BroadJump Sphere, where special services are hosted in order to provide functions such as auto update of software.

One role of the ESB architecture is to allow services to run in any sphere and interact with clients and services in the others without manual configuration. This makes it possible to individual spheres in or out of service without interrupting operations on any sphere.

The BroadJump Extensible Service Bus extends across hosts which run BroadJump client and server processes



3 Designed to scale

The number of client processes that may be connected at any one time is perhaps the single most important issue that the ESB architecture has to address. Building a software system that can support in excess of 250,000 active users means that every aspect of the architecture must give scalability top priority. Although systems on this scale are by no means common, a number of techniques have been shown to be effective.

The most reliable and manageable of these techniques is multiple dimensional partitioning. This is the separation of operational tasks into as many discrete and independent code paths as is practically possible. The ESB Base Services use several of these partitioning methods. The uses of each of these methods place certain limitations on designs that use them, frequently at the architectural level, so careful consideration of these issues in important.

The management of persistent data is frequently the most challenge aspect of large-scale systems. The choice of methods for storing such data depends on factors that may change as scale changes. For instance, commercial RDBMS systems can provide very high rates of access at reasonable cost at certain scales and for certain tasks. Using such a system for moderate load update processing of subscriber profile data is a good application. The load imposed by the management of dynamic data such as networking statistics when scaled to hundreds of thousands of users is a poor application. In such a case the rate of update would require a distributed architecture with many instances of the RDBMS, so the expense would likely outweigh the benefits of retaining the data.

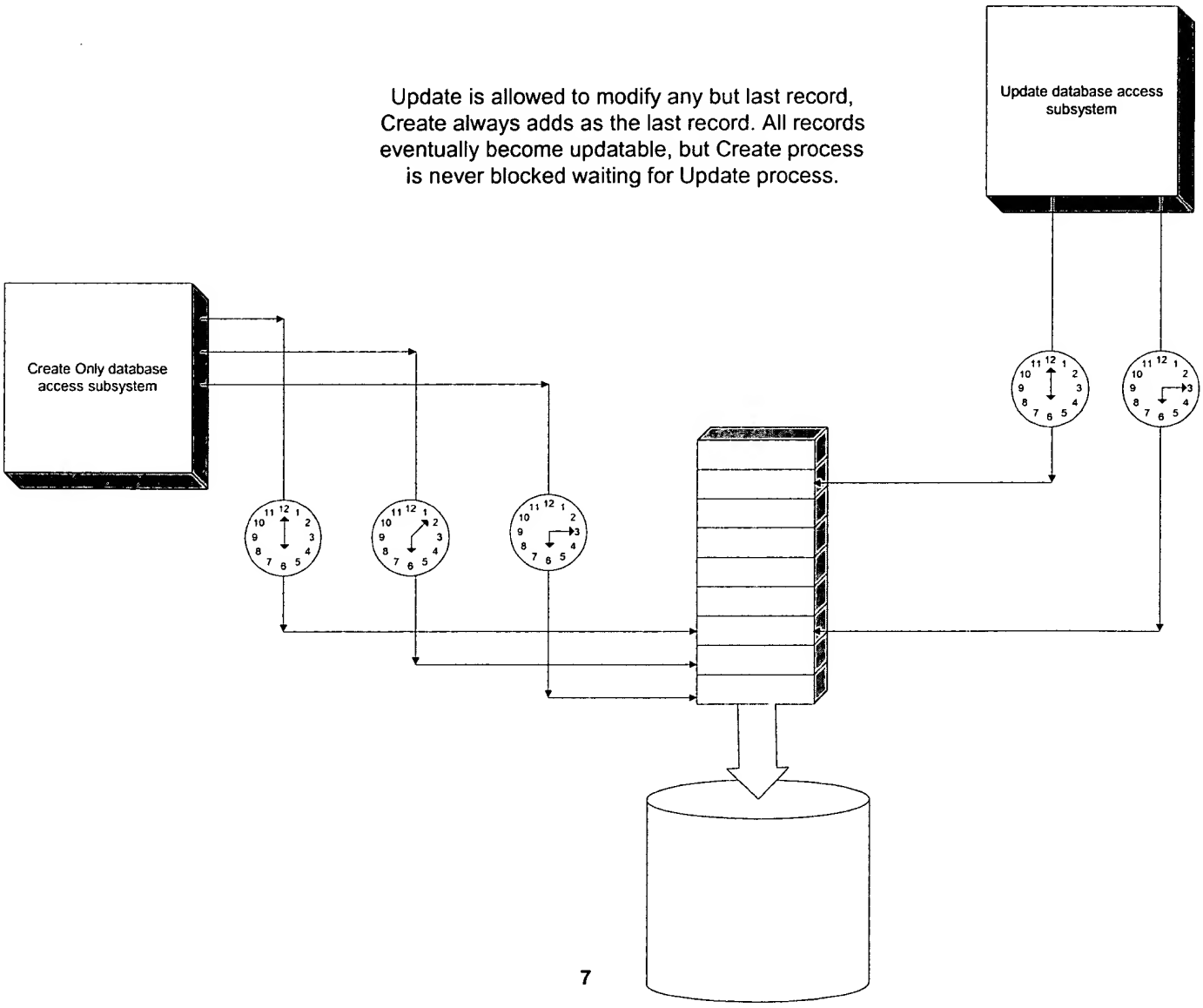
In consideration of these factors, this architecture makes no mandates about the nature of the database that stores any given class of data. All access to data is accomplished through interfaces that abstract the operations sufficiently that the underlying storage method can assume any form. The partitioning schemes are designed to be appropriate without regard to this form.

3.1 Functional partitioning

The method most widely used by ESB services is "functional" partitioning, which divides a task into discrete functions, and then separates the functions into independent services. This is a powerful technique that can handle many of the common service needs, but it has limitations.

One limitation is that it is sometimes difficult to separate functions. For example, it is sometimes not possible to use this technique for data management operations. In this class of problem necessary to query and update data within the context of a transaction that maintains ACID properties familiar in the OLTP world.

It is sometimes possible to use this technique for data management if there is a "create" operation that is distinct from "update" operations. For example, data that is created more or less constantly but updated only rarely can be jointly managed by two services, one that provides a high-speed create function and one that provides the necessarily slower update function. The update service must be able to allow the fact that the data is changing while the update takes place. This is possible mainly through careful database design.

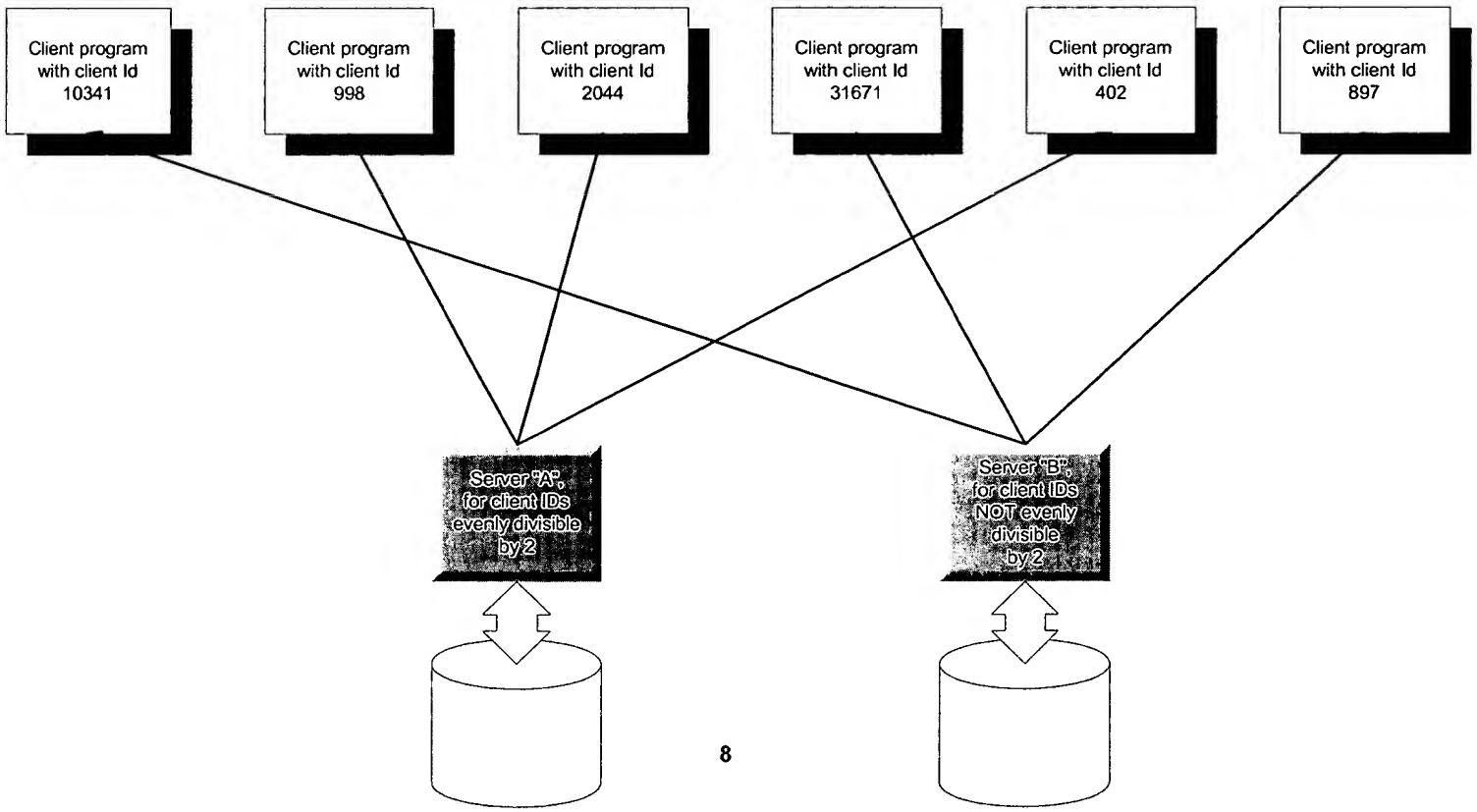


It is also possible to accomplish some functional partitioning in data management operations by separating out query only operations that can tolerate some delay in presenting the results of updates. For example, some classes of updates to a database can be queued for execution as a low priority task in order to devote maximum resources to delivering high-speed query access. In this model the queued data must be recoverable in order to tolerate service outages during the delay between en-queuing and actual update. There may also be a need to retry a failed lookup query operation in a way that includes recently updated data.

3.2 Partitioning by data dependent routing

Sometimes it is not possible to partition access to data via functional partitioning. This is often true, for example, in an application that allows users to purchase an item from a limited pool of such items, such as a physical inventory or a "first fifty to subscribe" pool. It is also true of mundane database operations such as updating a user profile, since the update might modify any item in the database. In some of these cases it is possible to provide the service in multiple server instances by making each server responsible for only a limited subset of the data, and then to direct the client to the appropriate server based on some key data in the request.

The Subscriber Profile update operation is a good example, since these data items are all tied to the primary key "Client-ID". This service is described in more detail below. As a simple example of how data dependent routing works, consider a Profile update service implemented by two server processes. Server "A" is responsible for entries that contain a Client-ID that ends in a number evenly divisible by two, and server "B" is responsible for the other half of all entries. When a client attempts to update an entry, it first checks the Client-ID field of the data it is reading or writing and performs the arithmetic required in order to determine which server is responsible. It then sends the request to that server.



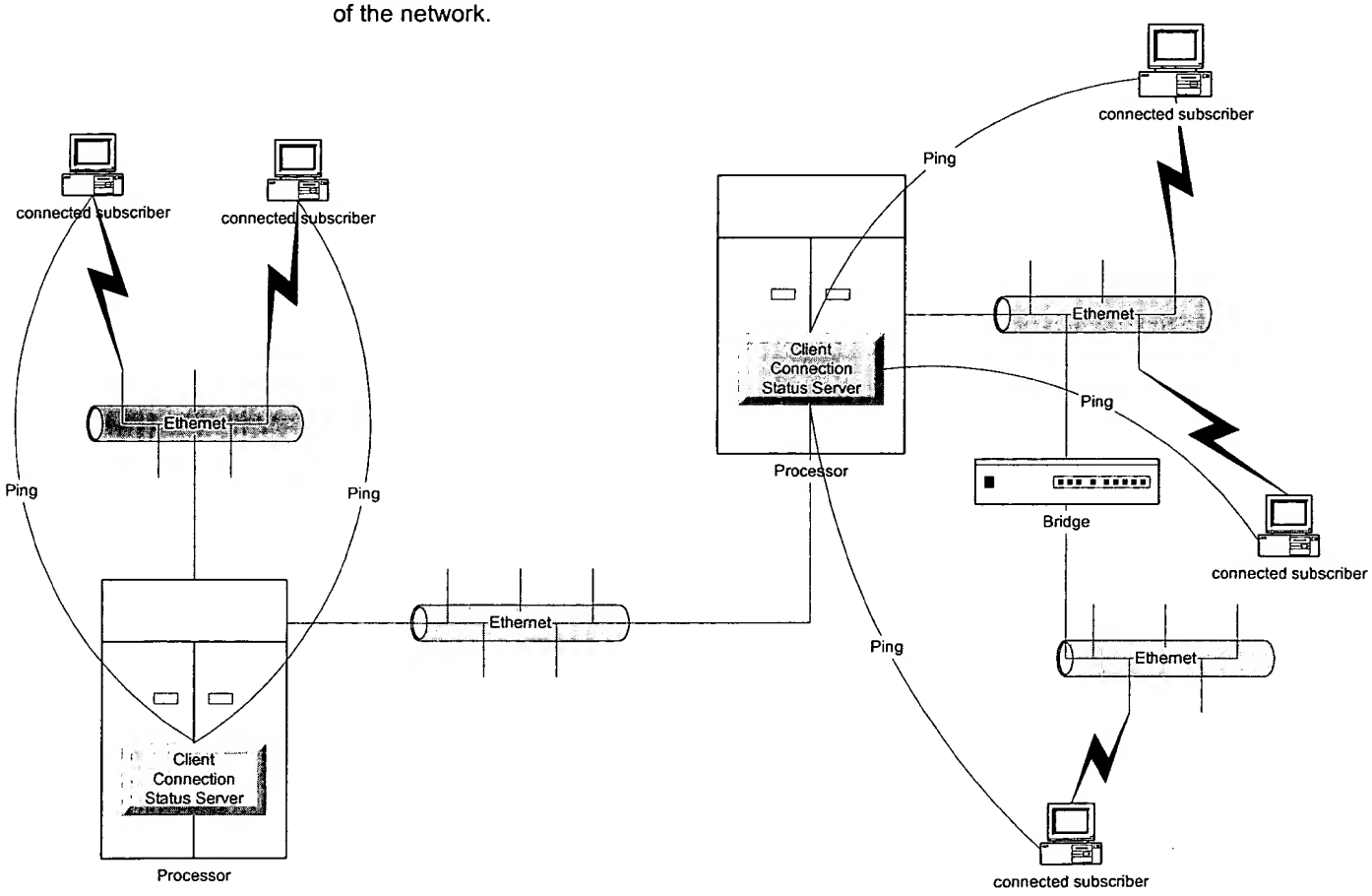
3.3 Partitioning by resource connection

It is frequently possible to predict some correlation between a service and its clients due to some third element that constitutes a shared resource and use this as a basis for partitioning.

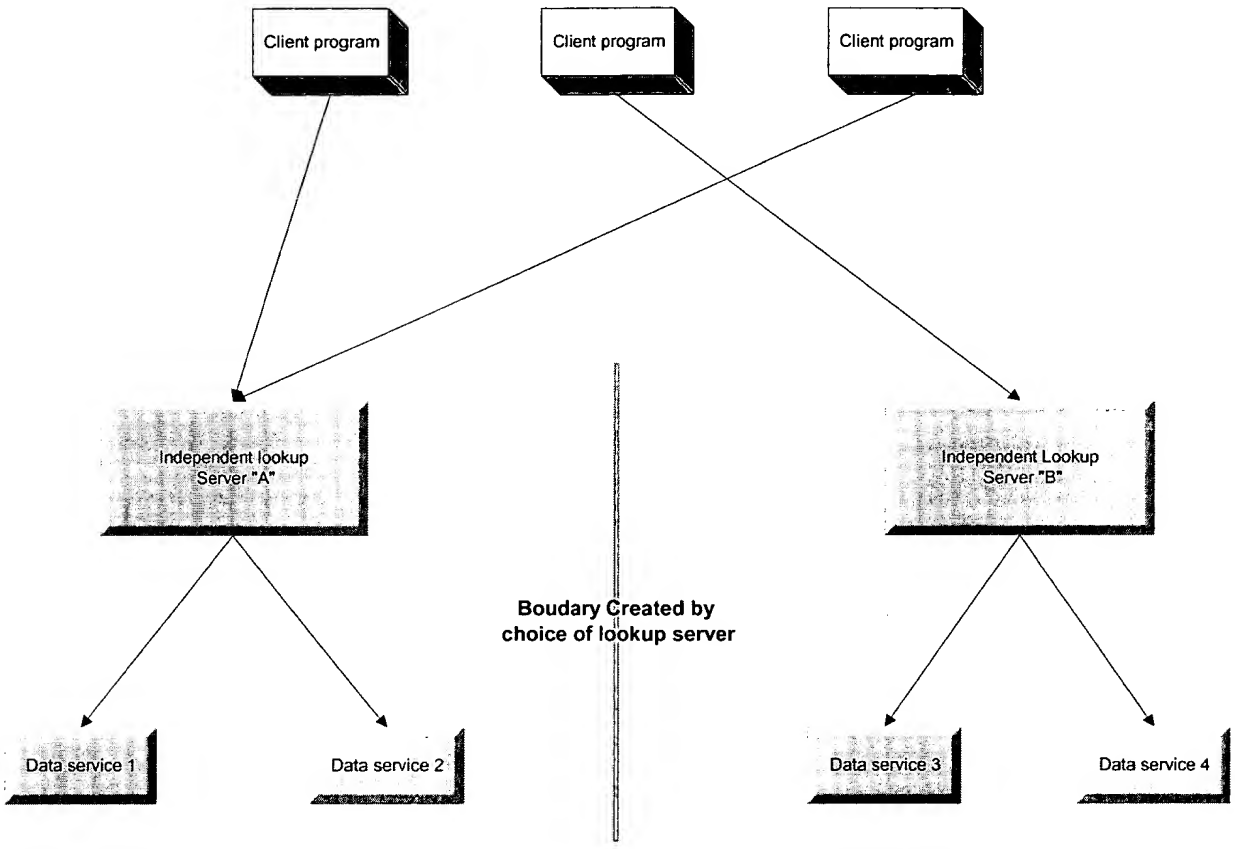
For example, this type of partitioning is used for the ESB connection monitoring service. This service monitors all processes that have connected to the ESB to detect cases where a process is no longer reachable but it still appears on the bus. This service does this partly by sending probe messages to the target client processes. Each target process runs on some machine in some sphere, and each machine has a position in the network topology.

The service is partitioned in to servers, each of which is responsible for part of the topology, and for the processes running on machines which share that part of the topology. This has benefits in limiting the load of the probes to the relevant part of the topology, and in increasing the total load capacity through partitioning.

Partioning by resource connection: Each Server is responsible for sending ping messages to the clients that must pass through the server's host on the way to the rest of the network.



A third example is found in the case where the client interaction with a group of servers follows a pattern where the client executes a series of service requests in a defined order before it reaches the result it needs. Finding a service through some sort of lookup operation is a typical example. It is often possible to partition the service that is first in that order as the resource connection, and to get the benefits of resource connection based partitioning in the subsequent calls. In other words, if each "lookup" server is associated with an independent set of the other services, then one of these service sets will be chosen implicitly when the lookup server is chosen.



Partitioning by resource connection, where resource is a server. The clients choice a lookup server by "coin flip". Since the lookup servers only access the data servers on their own sides of the boundary, the data servers are effectively partioned.

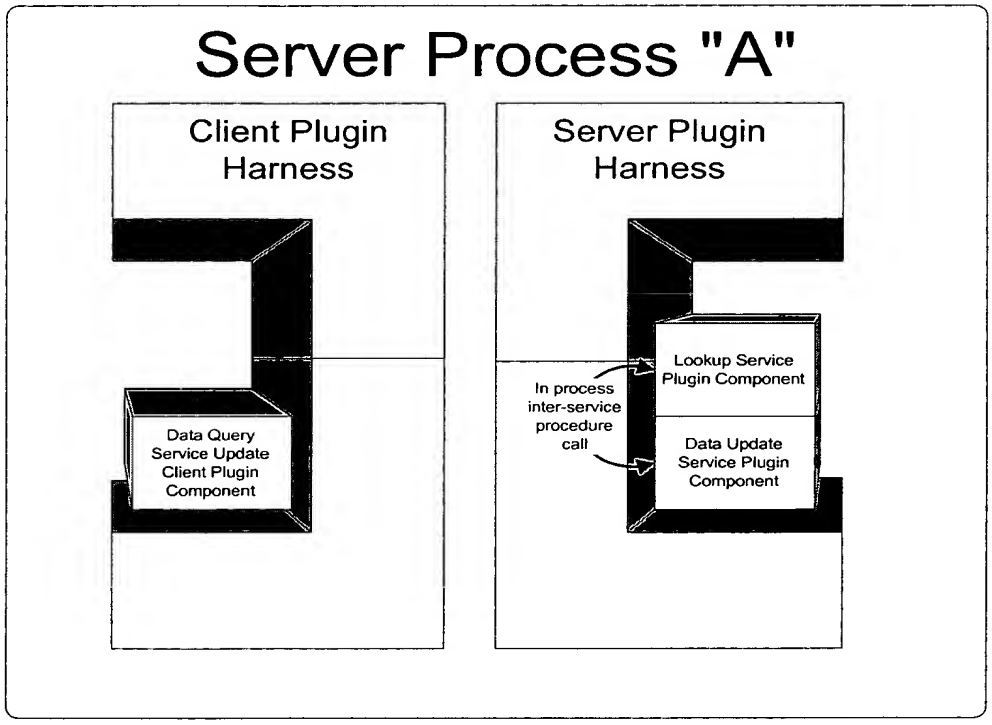
3.4 Partitioning by equivalency

In a small percentage of cases, a service may be of a type such that any server will do for any client request. An example of this is the service that provides Client-IDs for newly registered clients. The service creates an Id that is guaranteed to be unique across both time and space. This server can be replicated as broadly as necessary.

4 Designed for flexibility and extensibility

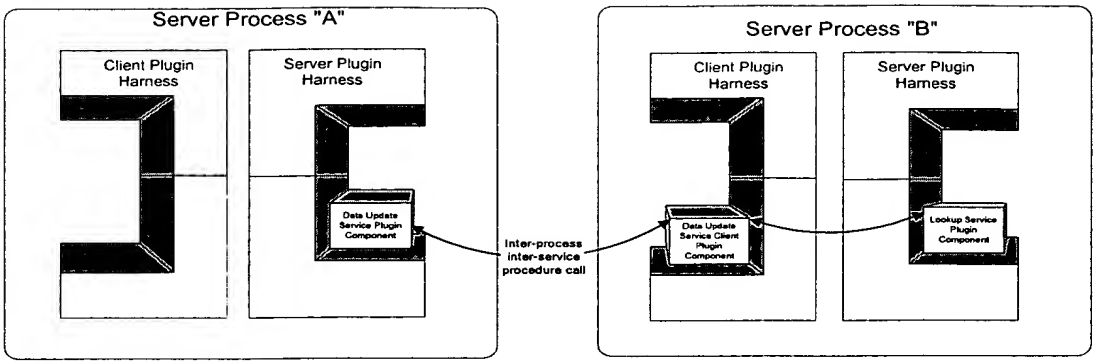
A dominant feature of the problem set that is addressed by BroadJump products is rapid change. The ISP experiences changes in the hardware and software technologies that must be supported, in the expectations of the subscribers, in the need for service enhancements to respond to competitive pressures. Many of these changes will directly translate to requirements for changes in BroadJump products.

In order to be able to respond to these changing requirements, the ESB architecture makes extensive use of plugin component technology in both client and server components. This allows the implementation architecture, i.e. which services run in which process on which machines, to be determined at runtime, and even to be changed without interruption of service.



The most obvious benefit of the runtime configuration feature is that it allows the service implementation to change in a way that tracks dynamic factors such as subsystem load changes. In this way it is possible to roll out a new service without serious risk of adversely affecting existing services. If there is an impact, the on-line reconfiguration feature allows the services to adapt to the new environment.

To understand the effect of this plugin approach on the service architecture, keep in mind as you read the rest of this document that all the services may be combined in one process, or may be distributed in complex ways simply by choosing which services to load into which server.

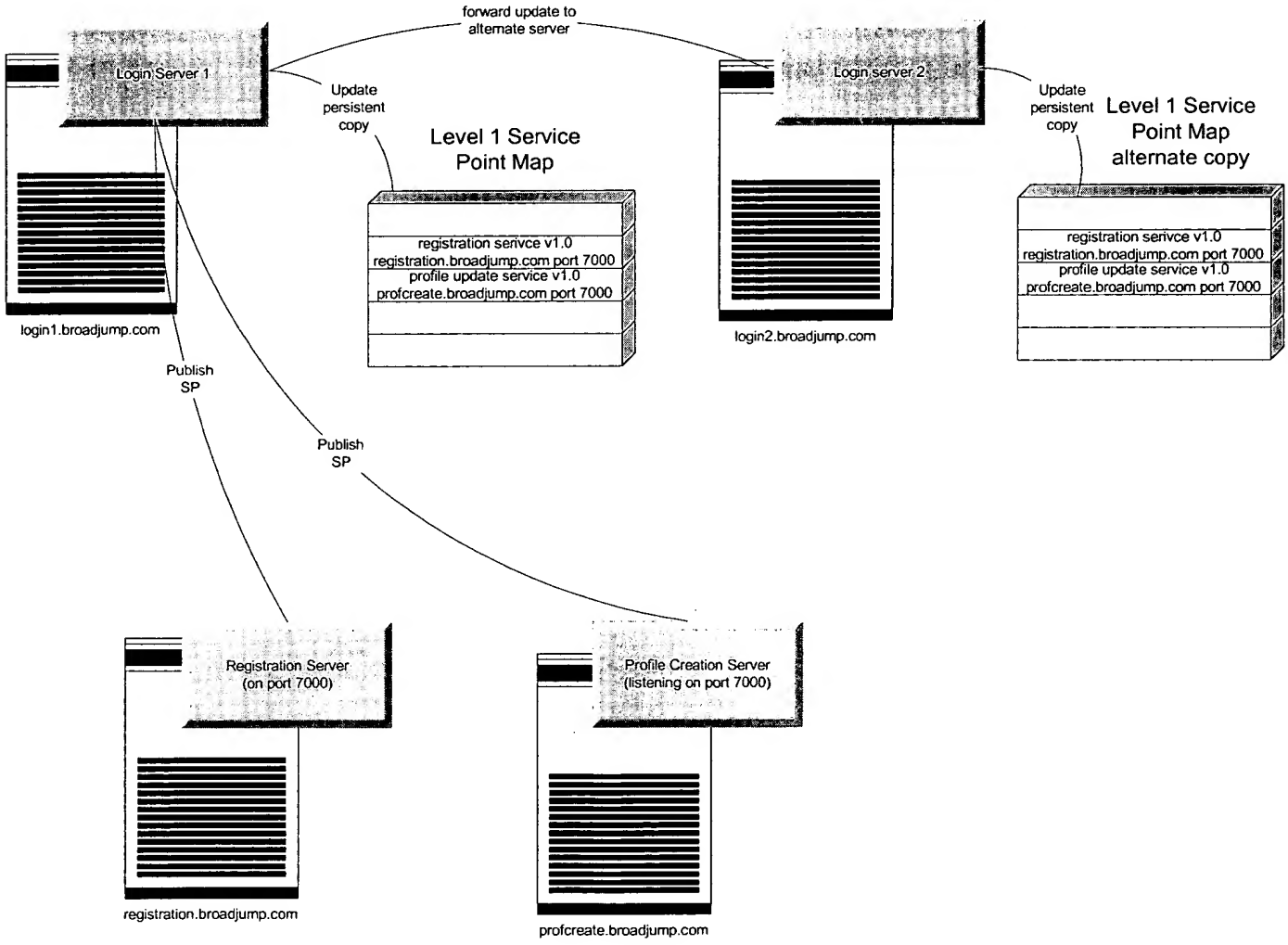


For more detail about the use of plugins and the adaptive configuration features, see Arch-x-200 "Service Access Protocol Architecture", Arch-x-300 "Server Plugin Architecture", and HLD-x-400 "Service Monitor Subsystem High Level Design".

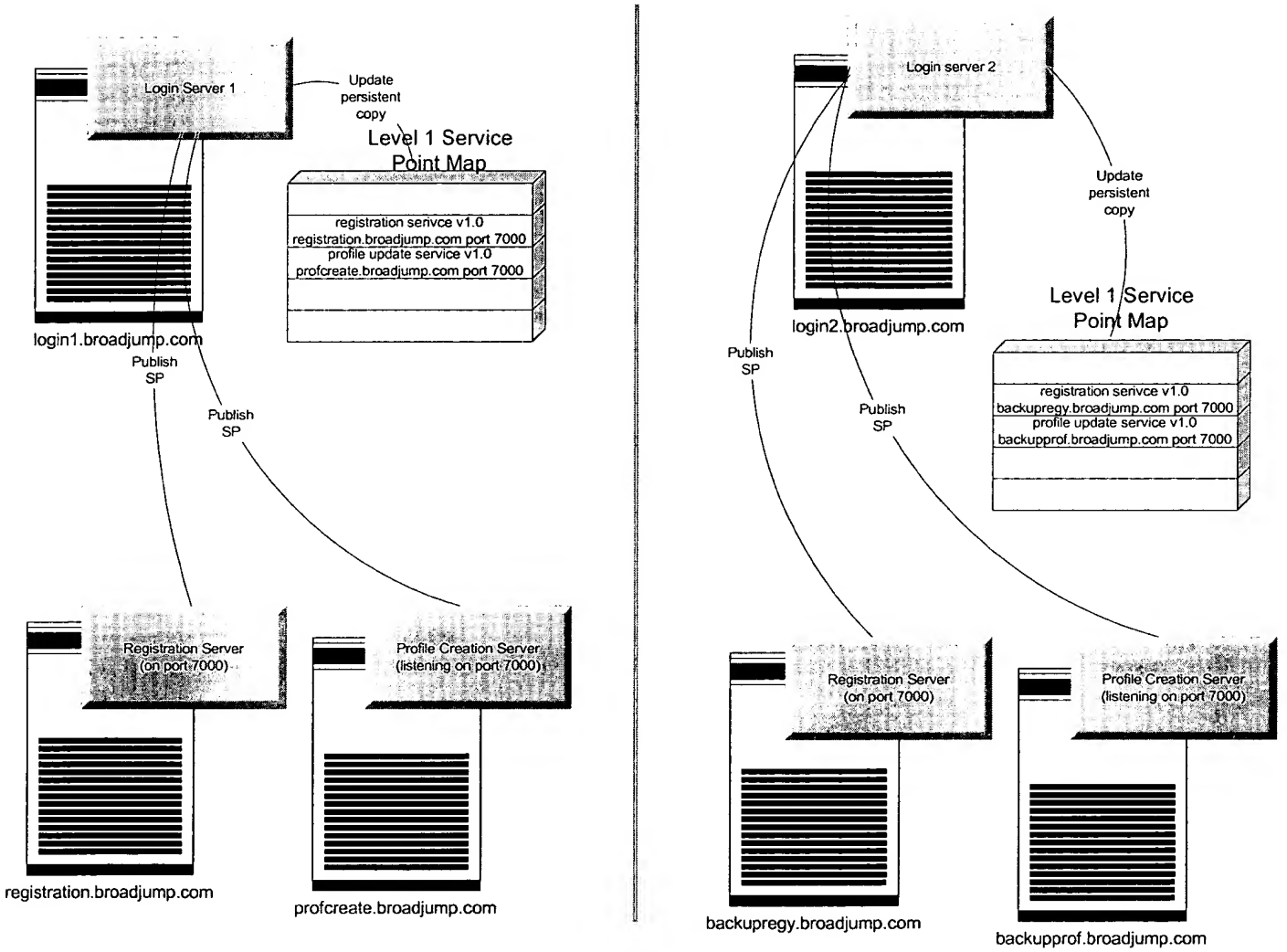
5 Service life cycle and ESB connection overview

The form of the service addressing information used throughout the ESB is called a Service Point Map (SPM). Each process in any connected sphere that implements a service has an entry in an SPM. The base ESB services have entries in the "Level 1" or "Top Level" SPM. This SPM is hosted and maintained by the login service. The Login service is the entry point into the ESB for most join operations, all base service joins and any non-first-time client joins. The communication addressing information is pre-defined for each sphere.

Several methods of pre-defining Login server location are supported, and the most likely is to use a port number that falls in a limit range of numbers, on a machine with a DNS name chosen from a set of "well-known" names. Methods are supported for defining multiple Login servers that share a single SPM, so the Login service can use equivalency partitioning.



Methods are also supported for defining multiple Login servers, each with their own SPM. This allows the



Login service and the services it lists to be partitioned by resource connection.

A service begins operations when the Process Control server on a machine becomes aware that it is supposed to start a server that implements that service. The PC server uses the appropriate OS dependent functions to start the Server Harness process and configure it to load the plugin for the requested service. The Server Harness calls the initialization code in the service plugin. See Arch-x-300 for more details on the Server Harness concept.

Each service constructs a Service Point Map entry for itself during this initialization. The SPM entry contains the service name, version number, and connection information for the specific instance, or server. A complete description of the SPM format can be found in Arch-x-200 "Service Access Control Protocol". The SPM entry is submitted to the Login service, at which time the service is said to be "active", implying that it is available to service requests.

When a server shuts down, it notifies the Login service with a list of the SPM entries that it has had active, and the Login service removes them from the Level 1 SPM. Entries will also be removed in the case that a server becomes unreachable, as a result of a message from the Service Control Manager.

For more detail see HLD-x-300 "Login Service High Level Design"

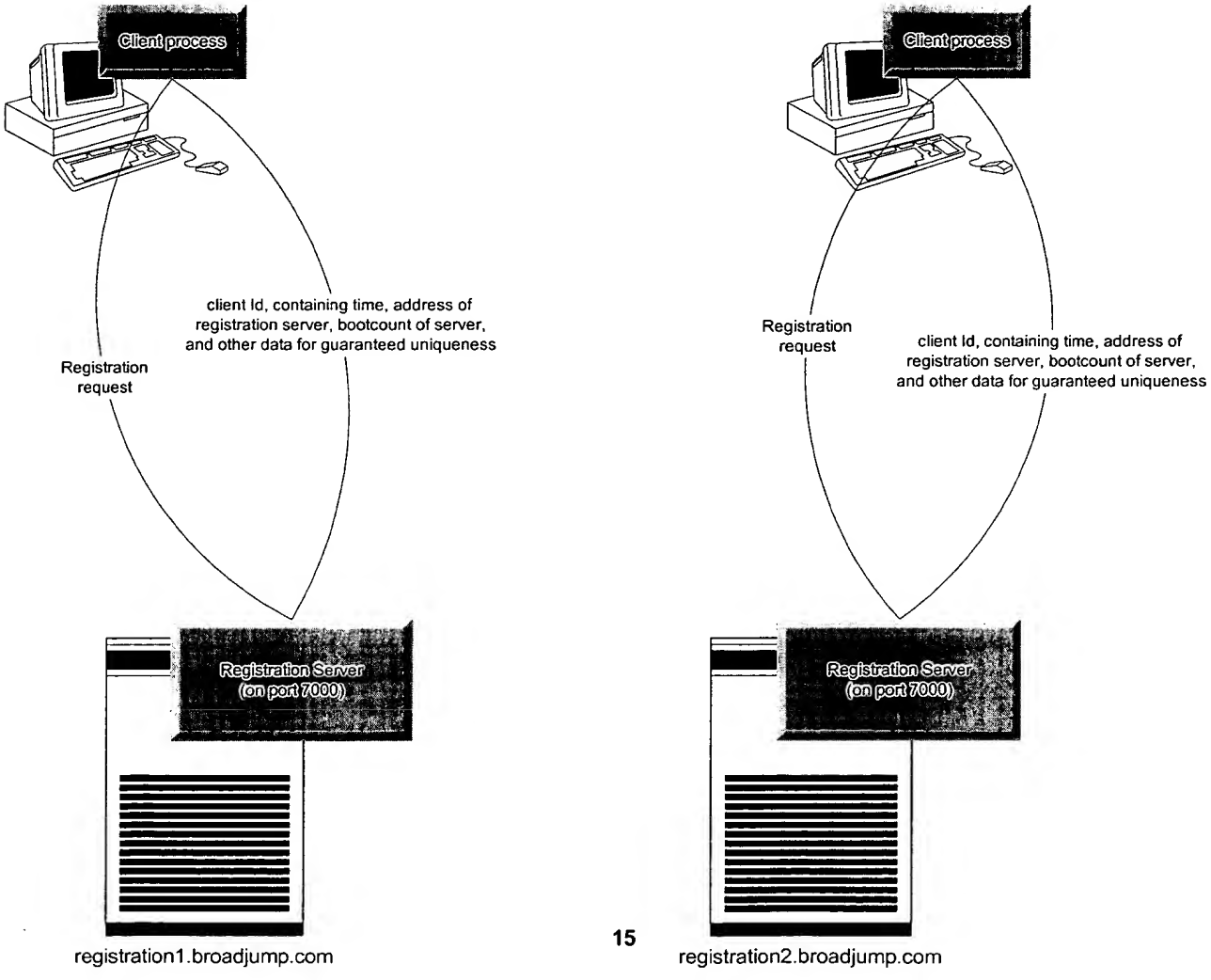
6 Client ESB connection life cycle overview

When a client connects to the ESB it follows one of two paths. If it has never connected previously, e.g. it is a new installation for a new subscriber, it contacts the Registration service. If it has connected previously, then saved local state will provide the information it needs to connect to the Login service.

6.1 Connecting through the Registration service

When a subscriber runs the BroadJump client for the first time, the client process establishes its place in the ESB by contacting the Registration service. The Registration service generates a Client-ID for the new client process. This value is unique in time and space, so multiple Registration servers can implement the Registration service without the need for shared state, thus enabling partitioning by equivalency.

The Registration service collects a current Level 1 SPM from the Login service and supplies this to the client. This enables the client to contact the services, effectively completing the connection process. The Registration service creates a new entry in the Subscriber Profile database (see below for more detail) using the Client-ID, so the client will be recognized on subsequent logins.



The Registration service follows the same rules as other services with regard to finding and registering with the Login service. As a result, the service can be partitioned under a Login service partitioned by resource model. The Subscriber Profile operations of the Registration service will happen inside the correct Login server partition.

The Registration supports the optional use of a subordinate service for retrieving data that was collect about the subscriber prior to the registration of this client instance. This Pre-Registration Data (PRD) service maintains entries that have been prepared during the process of arranging for installation at the subscriber's site. The data is searchable on any field, so the installation tool need only be provided with enough detail to uniquely identify the pre-registered subscriber, e.g. a work order number, a name and address, etc.

The Pre-Registration Data service also maintains a connection to the Subscriber Profile Query service so that it can support multiple client program initializations for a single subscriber. The two common reasons for multiple initializations are that the subscriber is using more than one machine, and when the client program is reinstalled without using a previously assigned Client-ID.

Once the registration process is complete, the Registration service also sends an update message to the Client Connection Monitoring service, which is described below. This message notifies the service that the Client-ID is "active".

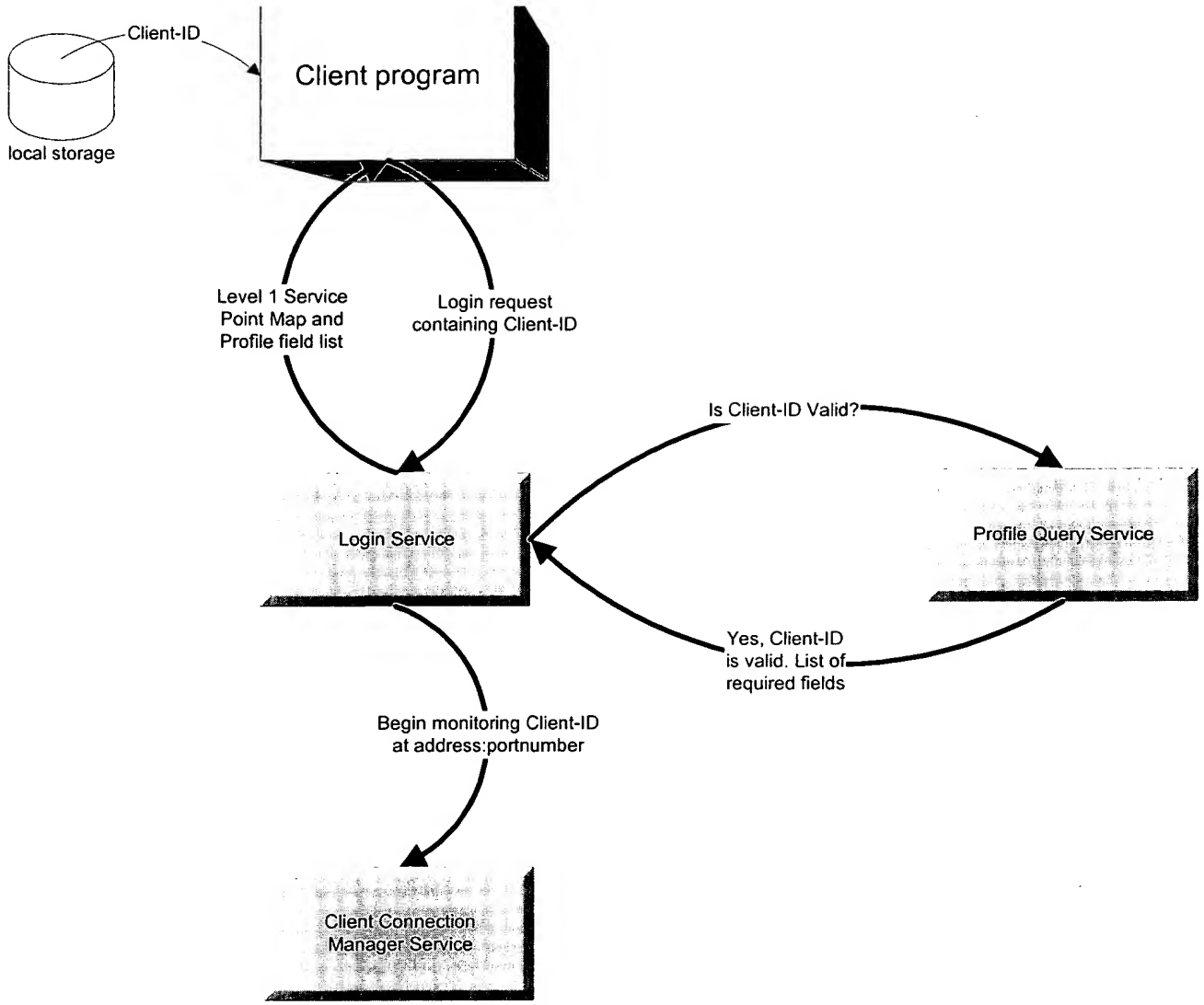
For more detail see HLD-x-400 "Registration Service High Level Design"

6.2 Connecting through the Login service

Once a client has received a Client-ID from the Registration service, it makes all subsequent connections to the ESB it by contacting the Login service, providing the Client-ID. The Login service verifies that the Client-ID is valid (described below in Subscriber Profile Query service description), and then replies to the client with a copy of the current Level 1 Service Point Map. The client is the able to connect to other services.

If the Login service is using a resource connection based partitioning scheme, then the SPM returned will include only services in the partition connected to the Login service that the client contacts.

The Login service also sends an update message to the Subscriber Connection Monitoring service, which is described below. This message notifies the service that the Client-ID is "active".



The reply from the Login service includes a list of the current required fields in the Subscriber Profile database. This allows the client process to determine whether it needs to collect additional data from the subscriber or the client machine.

6.3 Disconnecting from the ESB

When a client process completes a normal shutdown process, it sends a disconnect notification to the Client Connection Management service, which is described below. This service tracks the connection state of client processes.

Although a more symmetrical approach would be to send this notification to the Login service, this approach achieves better scalability through functional partitioning.

7 Subscriber Profile data management

In the ESB architecture, the term "Subscriber Profile" is used to label a set of information that is associated with a client process running on a machine in a client sphere. Note that this means that there may be more than one Subscriber Profile entry per actual ISP subscriber. The only data element that is guaranteed to be unique is the Client-ID, so this is used as the "primary" key for all operations on the data.

The definition is further limited to data that is relatively static in nature. For example, this data contains information about which version of the various client plugins are currently loaded into the client process, but it does not contain information about the status of messages originating at or addressed to the client process. For a full description of the Subscriber Profile data format, see HLD-x-500 "Subscriber Profile Manager component".

The management of this data is a key design point in the scalability of the ESB, since each connection of a client process to the ESB will create at least one query on this data, and possibly one or more update operations.

The query operation is part of the login process and is a result of a request from the Login service. This query is a search to ensure that the client process has presented a valid Client-ID number. A Client-ID number is defined as valid only if it appears in the Subscriber Profile database.

Registration of new clients accesses this data in order to create a new entry for the new Client-ID. Although the only field in an entry that is required architecturally is this Client-ID, the practical requirements of the various services and the specifics of a given deployment will add fields to the minimum set. The details of the process of collecting the data to fill these required fields will vary depending on the nature of the data.

For example, if an ISP chooses to have the data entered by the customer service representative at the time that the client places the order, then it will be supplied to the registration service via the pre-registration database. One the other hand, the ISP may choose to have the technician complete all required fields during the installation. In any event, the registration client plugin will not complete the registration process until all the required fields are supplied. An adaptive programming method is used to supply the list of required fields to the registration client program at run time, so this field list may change at any time.

Both the Login and Registration services must be prepared to meet extraordinary peak demands, so each needs high speed access to the data. Since high-speed query and high-speed update operations are somewhat mutually exclusive with large data sets, the Subscriber Profile database is implemented as a set of three services partitioned functionally. They are the Subscriber Profile Create service, the Subscriber Profile Update service, and the Subscriber Profile Query service.

7.1 Subscriber Profile Creation service

The Subscriber Profile Creation (SPC) service uses a data set that is implemented as a queue that is externally accessible to reader processes and which is normally accessed in FIFO order. The service does not return a reply to the "create" request until it has completed the en-queue operation. This ensures that at the completion of the registration request the Client-ID is in persistent storage. Under these conditions abnormal process termination or machine failure will not cause loss of the Client-ID.

The service can be partitioned into multiple servers using equivalency. This is possible because the primary key for Subscriber Profile data is the Client-ID, which is guaranteed to be unique. The Registration service is the only client of the SPC and it will never submit records with duplicate keys.

The SPC service includes a Queue Reader sub-component that operates as a service, i.e. there is a client plugin associated with it. This plugin is used by remote processes for the purpose of reading and de-queuing messages. Reader processes use the initialization function to notify the service of their locations in the deployment architecture, i.e. the name of the host on which the caller is running and its process id.

If the caller is on a different machine, then subsequent operations take place via calls to the Queue Reader service plugin that shares the server process with the SPC. In this way all the locking operations that serialize access to queue control data structures take place within a single process.

If the caller is on the same machine but in a different process, then the Queue Reader service in the SPC process and the Queue Reader client plugin in the calling process cooperate to establish a logical

semaphore via operating system methods. This semaphore is then used to provide the inter-process locking semantics and the caller can operate on the queue directly.

If the caller is in the same process as the SPC, then locking takes place in memory. The second alternative, different process on a single machine, is generally the best alternative if symmetric multi-processing machines are used as hosts, offering the best scalability through partitioning and the best performance by avoiding network operations.

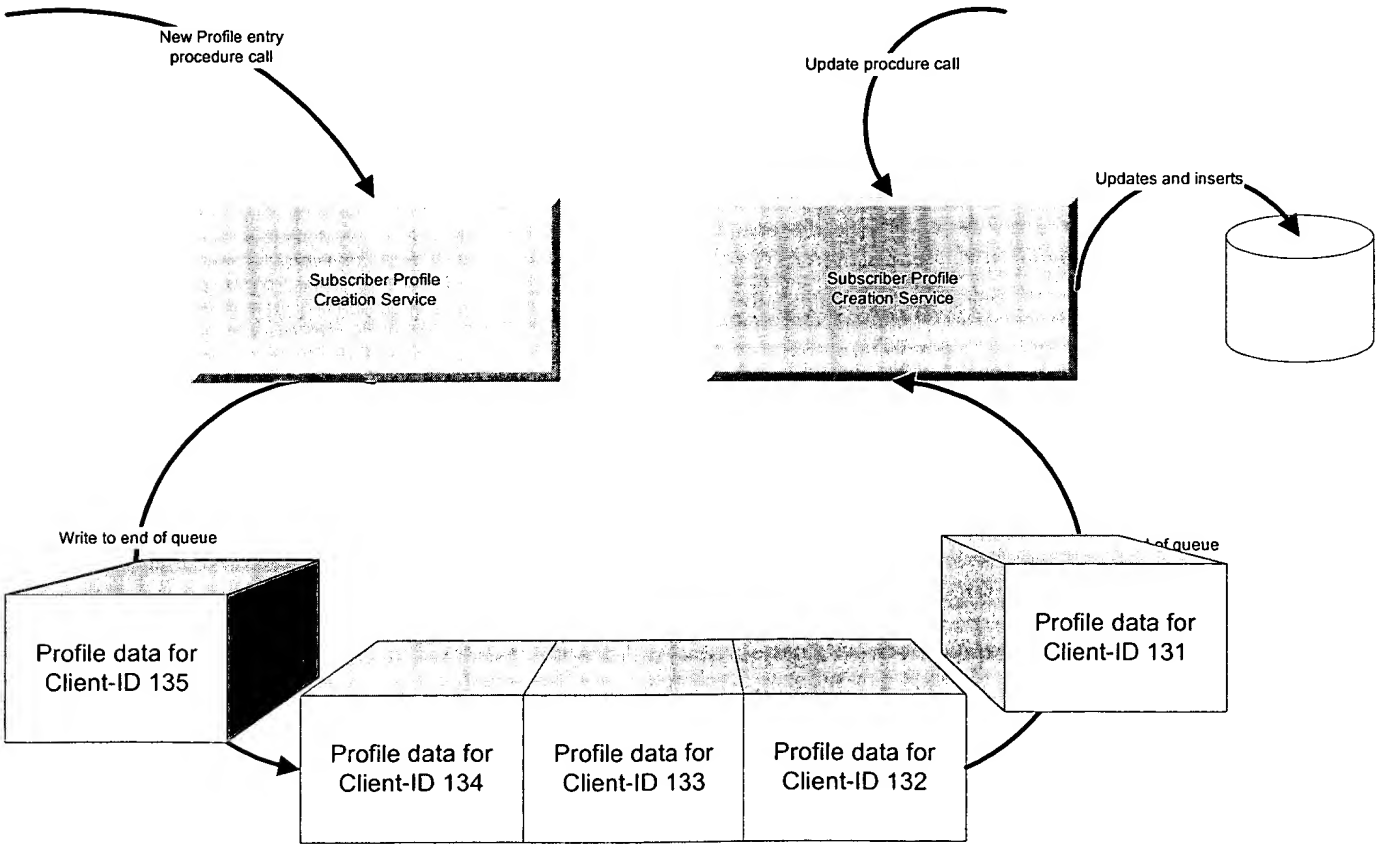
The locking operations for en-queuing and de-queuing are generally not mutually exclusive, except when the head and the tail of the queue approach each other.

For more detail see HLD-x-600 "Persistent Queue Manager component High Level Design" and HLD-x-700 "Subscriber Profile Creation service High Level Design"

7.2 Subscriber Profile Update service

The Subscriber Profile Update (SPU) service has two roles in managing Subscriber Profile data. The first is to accept requests to modify existing entries in the database, and the second is to act as the Queue Reader for the queue or queues created by the SPC service.

The handling of update requests is usually straightforward. The SPU simply searches the database for an entry that matches the Client-ID provided in the request, and then modifies the entry accordingly. The changes are logged to disk so as to make the service recoverable in the event of failure.



The SPU service may be partitioned into servers using data dependent routing, since the entries are always referenced by the primary key, the Client-ID.

In the Queue Reader role the SPU service accesses each of the queues written by the SPC service and de-queues entries. These entries are then placed in the database. If data dependent routing partitioning is active, then each SPU server will collect only entries which match its own data range rules.

The actual order of de-queue and update is reversed compared to the conceptual description above. The data is read from the queue, placed in the database using a "create or update" operation, and then de-queued. This removes the need to complete both steps in a single transaction context, which would be a distributed transaction if the SPC and SPU services were plugged in to different processes. Using this technique, failure recovery is automatic as long as the queue has not been corrupted.

Any process that performs operations on the queue acquires a distributed lock. There is a single lock for A pessimistic algorithm is used, such that the lock is acquired if any.The implementation of this locking method comes in forms suitable for processes co-resident on a single machine, and for processes on different machines. The former is obviously far faster and more efficient, so this deployment choice is highly recommended.

The en-queue operation always sets a lock flag for the duration of the operation, and the de-queue operation waits to de-queue the last entry until this flag is cleared.

Another function the SPU provides in the Queue Reader role is to search for a specific entry in the queues upon request. If the entry is found it is transferred to the database and de-queued, and the reply to the request returns the entry.

The database operations use a logical layout that makes it possible to validate a Client-ID even if the associated record is locked for update.

For more detail see HLD-x-900 "Subscriber Profile Update service High Level Design" and HLD-x-1000 "Subscriber Profile Database Interface High Level Design"

7.3 Subscriber Profile Query service

The Subscriber Profile Query service is relatively simple. It supports two database lookup functions, one that reports whether a given Client-ID is in the database, and another that returns the contents of the entry for a given Client-ID. This separation is used to enable the fasted possible response time to the Login service, which needs only the verification function.

If a Client-ID is not found in the database, the SPQ service contacts the SPU service to find out if the entry is in the Creation queues. If the entry does not exist in either of these locations, the SPQ checks the time value embedded in the Client-ID and uses this to chose an action.

If the Client-ID was issued recently, then the SPQ signals an administrative alert indicating that a probable service hang exists somewhere in the path, and returns a related error condition to the requester. If the Client-ID issue time was not recent, the SPQ signals an administrative alert indicating that either a database corruption or an unauthorized access attempt is suspected. A related error condition is returned to the requester.

In either event, the invalid Client-ID is stored in a special table that is used to detect denial of service attacks.

For more information, see HLD-x-800 "Subscriber Profile Query service Level Design"

8 Client Connection Monitoring service

The client connection life cycle described above explains the actions that delineate the presence or absence of a client on the ESB. When the Login service accepts a client login request it sends notification to the Client Connection Monitoring (CCM) service. This service monitors the life cycle of a client connection from that point in time until the client disconnects by sending a disconnect request to the CCM. It uses two component services to accomplish this, the Client Connection Status (CCS) service and the Client Connection Directory (CCD) service.

The Client Connection Status service implements a "keep alive" feature by sending a probe signal to each client periodically. The interval used is a runtime configurable option that can vary by Client-ID. The probe signal can return information that is collected from the client machine, such as network statistics. This is a runtime configurable option that applies to all monitored clients. This data is stored in either by latest value or by cumulative value according to the data type.

Callers to the CCS service can use two methods to gather information about the status of a client, polling and subscription. In the polling method the caller simply sends a request for the status of the client, and receives a "connected" or "not-connected" indication. If the client is connected, the caller also receives the current monitored data for that client. In the subscription method the call registers itself as a "subscriber" to connection status for a given client. When the client disconnects the caller receives a message containing the monitored data for the client. All client connection and disconnection events are logged to a database.

Ideally the CCS is partitioned into servers that are placed at the outermost extremity of the network topology that is closet to the client. This resource connection based partitioning method ensures that the probe transmissions will each effect the smallest possible part of the network.

The Client Connection Directory service controls and tracks the association between CCS servers and Client-IDs. The CCD provides an API that allows the nature of a new connection to be examined by logic that understands the network topology and which assigns the monitoring of the client to the appropriate CCS server. This assignment is recorded in forms that allow searching by a number of hierarchical patterns, such as network topology, connection duration, Client-ID, etc.

When a process wants to access or modify status for a client, it must first contact the CCD and search for the association between the Client-ID and a CCS server. It may then contact the CCS server and make the request. Note that the address information received in this process is a normal Service Point Map.

It is expected that the CCD will commonly be deployed as a service running in the same server process as an instance of the Central Directory service.

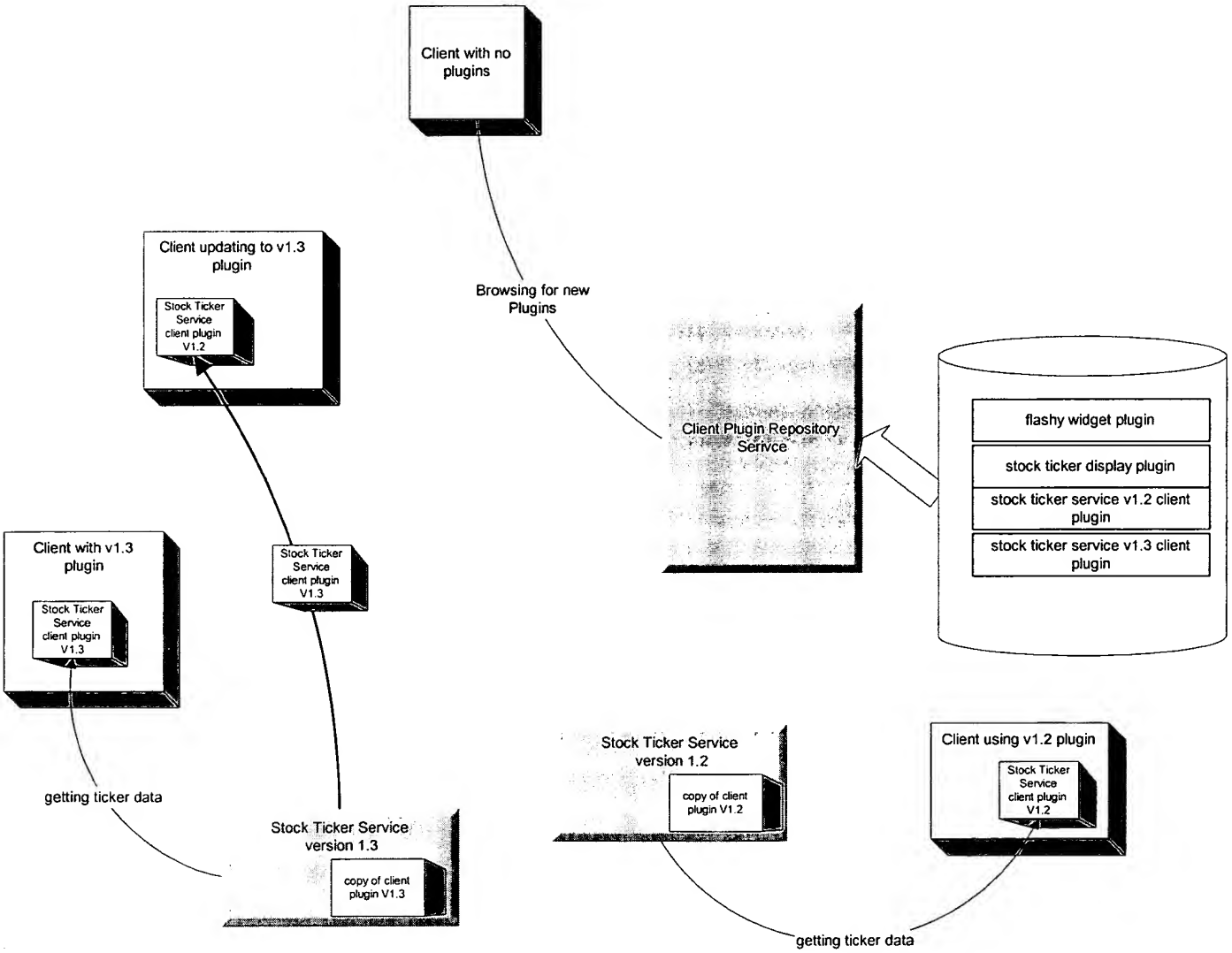
For more information, see HLD-x-1100 "Client Connection Monitoring service High Level Design"

9 Client Plugin Management service

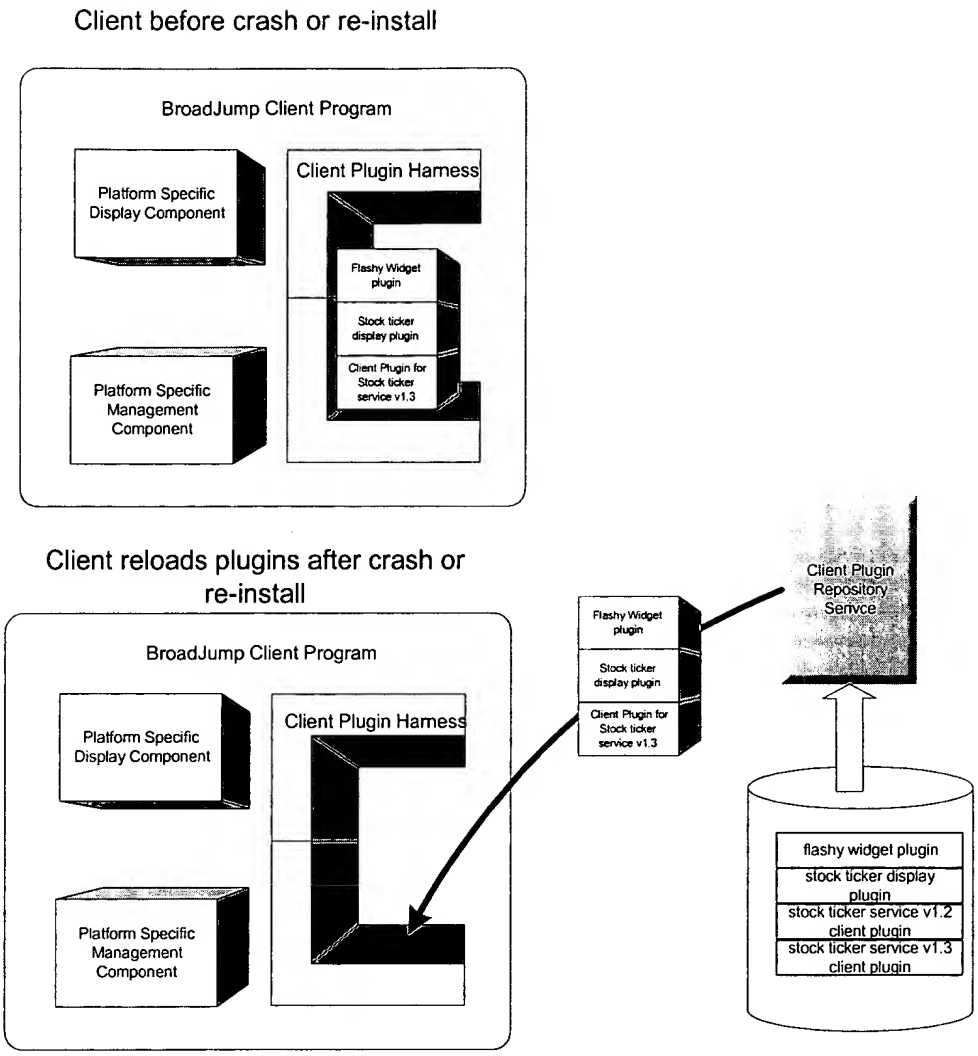
The Client Plugin Management service (CPM) ensures that individual plugin modules in client processes remain up to date. Some updating of client plugins will take place automatically, some will take place upon subscriber request, and some on administrative request The definition of "up to date" varies by plugin type.

For example, the client interfaces for ESB base services are required to be of the same version as the service that is being used. As a counter example, a new service connection plugin for a non-essential service might be considered up to date as long as it supports version X of the interface or better. Another example is a new version of a user interface component that the subscriber may choose to load or not. Many other definitions of "up to date" can be imagined.

All client plugins are stored in the Client Plugin Repository (CPR) service. This is a simple database that is optimized for read operations, as the data is fairly static in nature. The plugins may be indexed by name and version. The CPR also supports a browsing feature that presents the available plugins in a directory style hierarchical structure. Multiple references to a single plugin are permitted in this directory structure, so the plugins can be located in a task specific fashion.



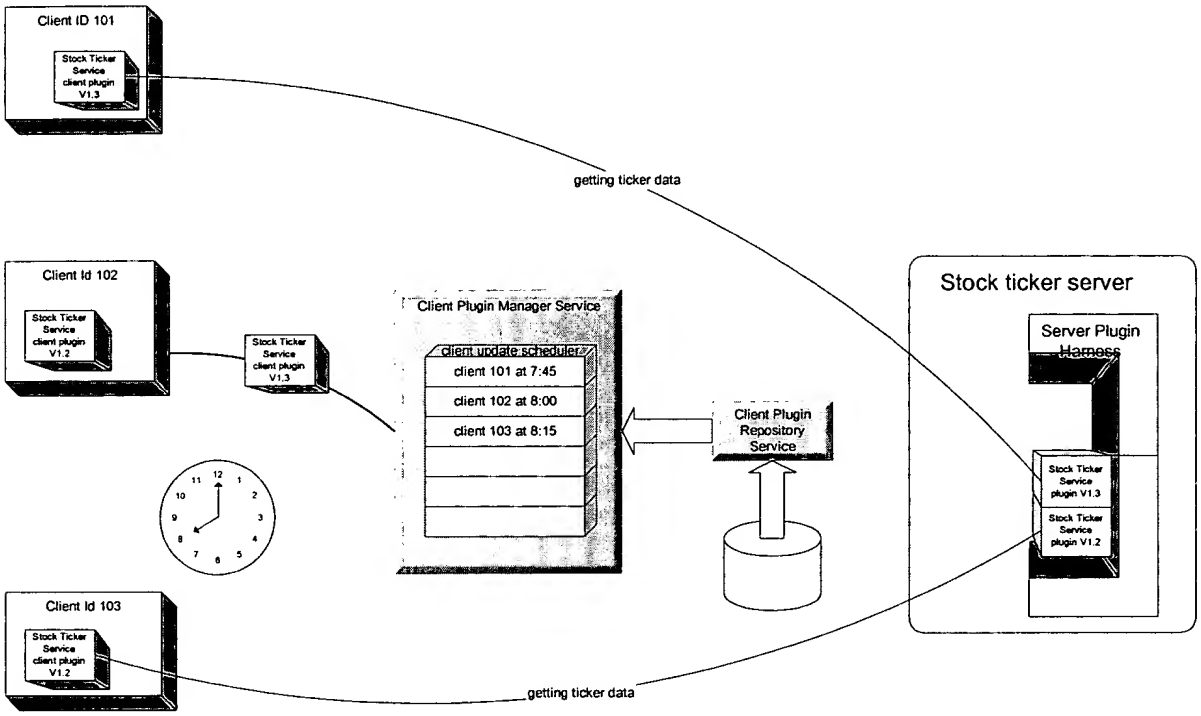
Plugins that provide connections to services are typically managed by the services themselves. Each service instance keeps a copy of the client plugin versions that it supports. When a client first contacts a service a query protocol determines whether the client needs to update its plugin for that service. If so, the service provides the plugin to the client directly, and the client performs the update immediately.



Since this method of updating a client plugin is predicated on the ability of the client to contact a service, and since that ability is provided by the client plugin associated with that service, there is a "first-load" process. When the client process becomes aware that it needs a client plugin for a service, it loads the latest version that is available in the CPR.

The Subscriber Profile service stores a list of all the client plugins that have been loaded by each client process. This is used by the CPM service to execute scripted update logic in order to determine client update schedules. It is also used to reload a client process with its previous state in the event that the client machine has lost its locally stored state. The entry for a given client also stores data about availability of new plugins and updated plugins.

The CPM service provides administrative access to the CPR for the installation of new plugins, and to the Subscriber Profile data regarding plugins for planning and execution of updates. The client process always contains a CPM client plugin, which is able to accept update alerts from the CPM. A scripted scheduling mechanism allows the administrative user to schedule updates to occur by user.

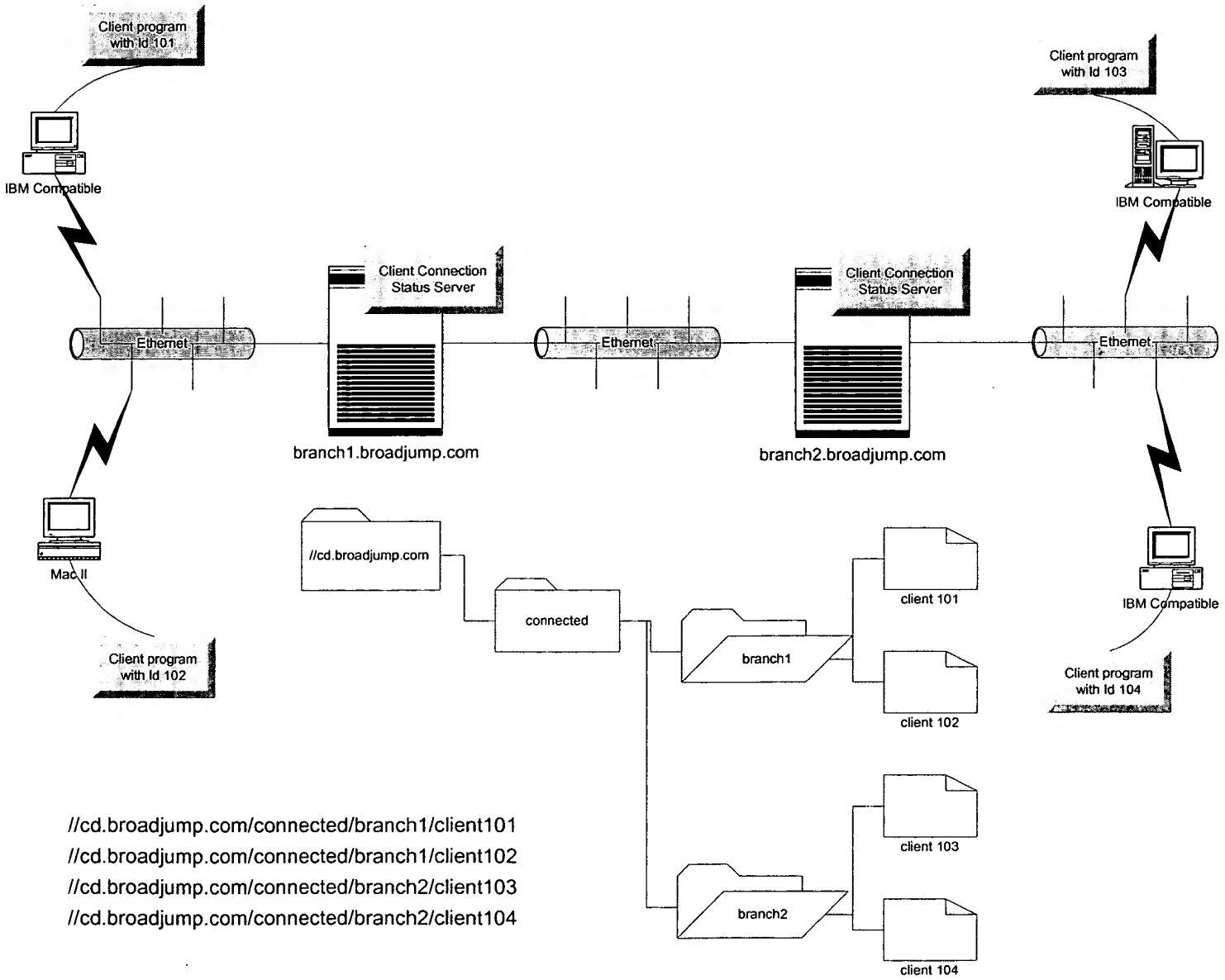


For more information, see HLD-x-1200 "Client Plugin Management Service High Level Design".

10 Central Directory service

The Central Directory service presents a unified interface for searching and updating data that is managed by ESB base services. It implements the view portion of a Model View Controller application architecture in which the other services and their databases play the roles of model and controller. All the data that is available in a sphere is presented in a familiar hierarchical directory structure.

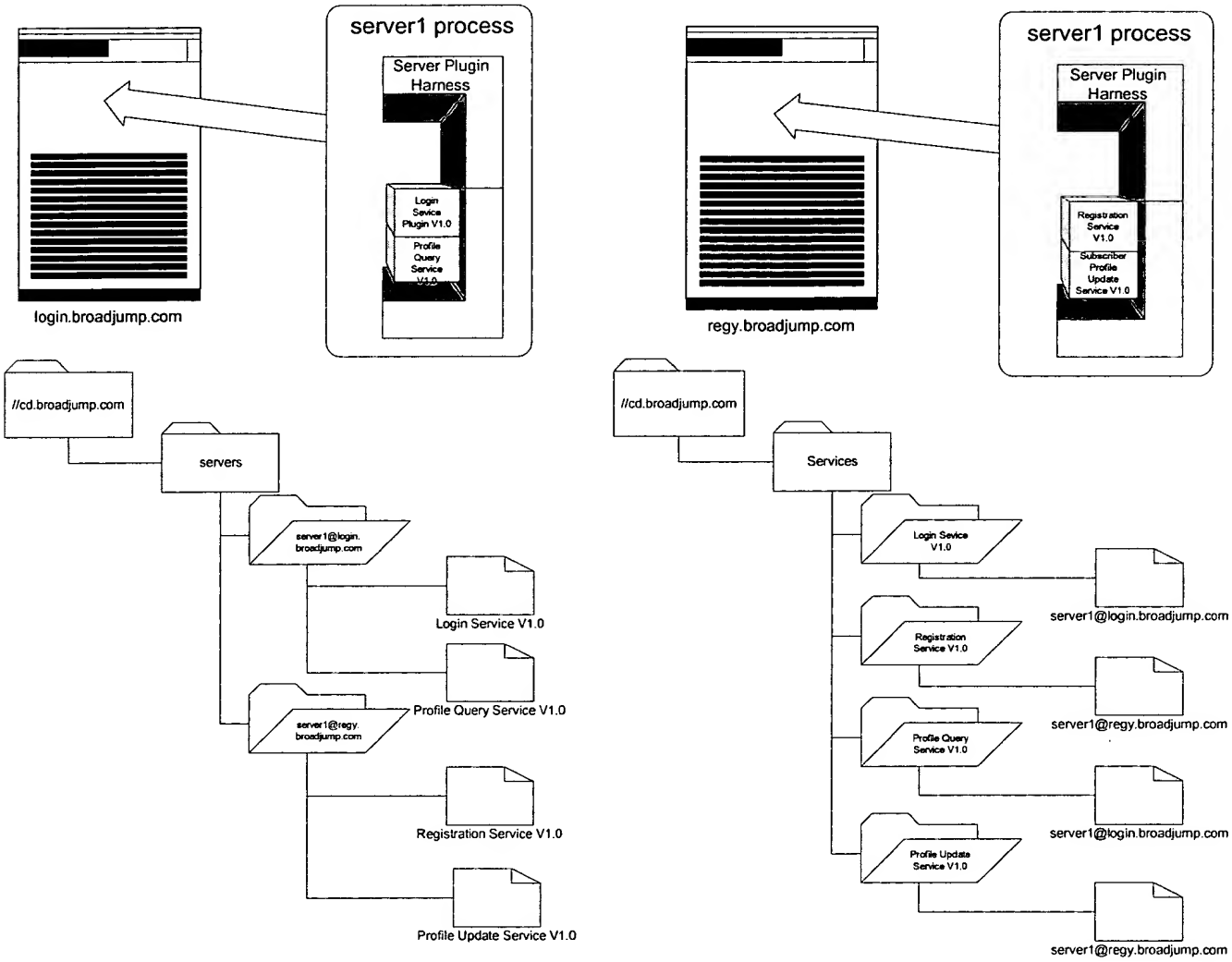
The root of the directory structure contains a number of predefined entries, each of which represents mode of viewing data.



//cd.broadjump.com/connected/branch1/client101
//cd.broadjump.com/connected/branch1/client102
//cd.broadjump.com/connected/branch2/client103
//cd.broadjump.com/connected/branch2/client104

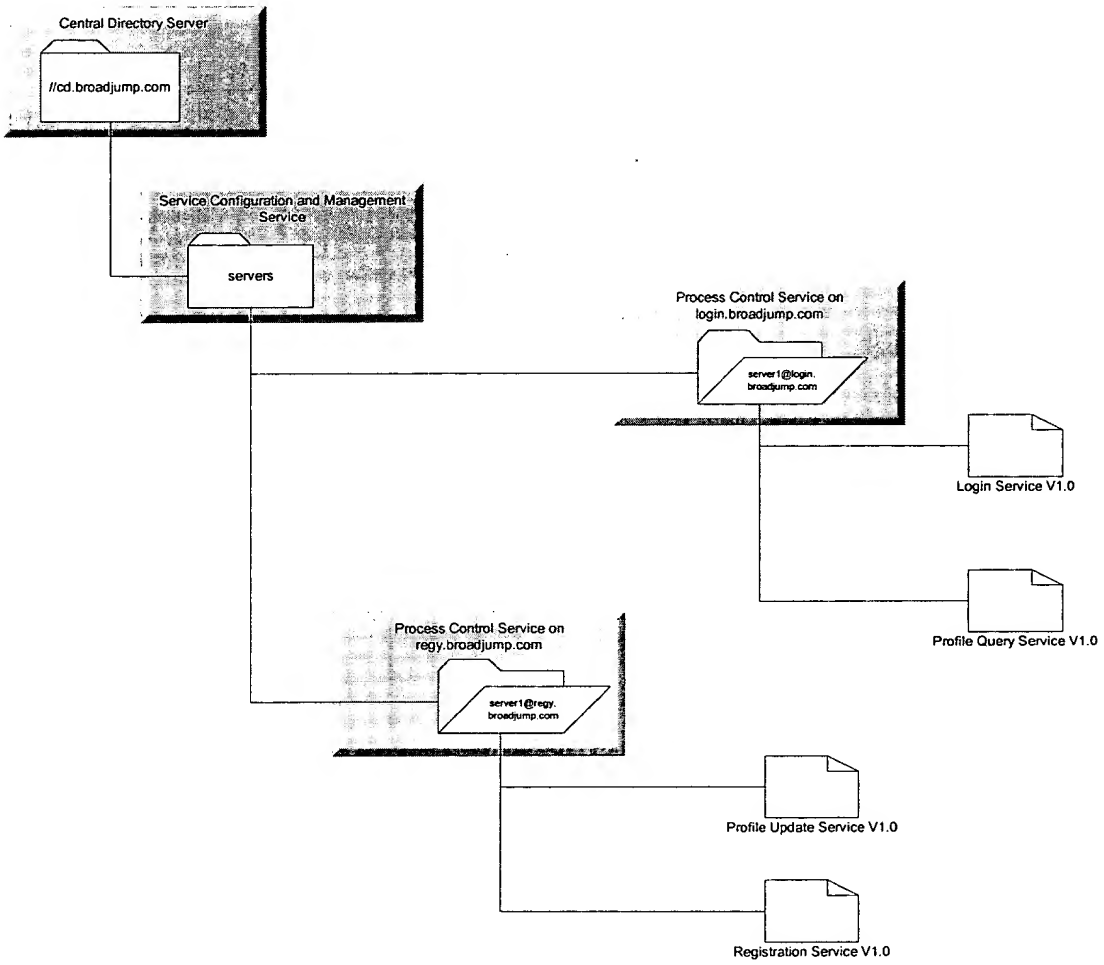
For example, there is an entry called "connected". This is the root of a hierarchy that shows elements of the network topology and the clients that are connected to them. This data is mostly provided by the Client Connection Management service.

As another example, the service deployment details are represented in two ways each with its own root. The "services" root leads to trees that end in server names and locations. The "servers" root shows locations, then processes, and then services. These are two views of the same data, which is managed by the Service Configuration Management service.



Subscriber Profile data is presented via many views, many of them diverging at the root level. For example, the geographical location data in the Subscriber Profiles is used to create a tree of increasing specificity. Common demographic distinctions are used as well, so that a root called "age" exists, which extends to a tree with branches of increasing specificity. Individual clients are identified by Client-ID, and all the other Subscriber Profile elements are presented as properties.

When any of the data that is represented is modifiable, the Central Directory service supports a "junction", a transparent connection to the service that is responsible for the data. For example, in order to add a service to the list of plugins that a particular server will load, the Central Directory "new" or "move" commands can be used. These will be translated into service requests that are then sent to the Service Configuration Management service.



These "junctions" do not appear do be any different from any other element of a path, except that their properties contain values which specify the name, location and type of the server that is responsible for the junction.

The Central Directory uses a distinction between "containers" and "objects" as two classes of entries. Object entries are items such as service processes, Client-IDs, etc. Containers are structural entries that group objects, roughly equivalent to directories in a file system, except that deleting a container does not necessarily remove the objects it contains. The correlation is strong enough, however, that the term "directory" is used interchangeably with container.

Object and container entries have properties associated with them. There are some that are associated with all entries, some that only objects or only containers have, and some which are specific to particular types of containers or objects.

For example, all entries have an associated "access control list" property. This is a security mechanism that allows administrators to grant and deny various privileges, such as read, write, delete, etc. Container entries have "parent" and "child" properties that are used to implement the hierarchical structure, and which are not associated with objects. Objects have a "plugin" property that specifies the name and version of the plugin that allows manipulation of the object, if such a plugin exists. Objects of the "connected client" type have properties that contain the networking statistics for the client, priorities that are not associated with client plugin entries.

Objects are frequently contained in more than one container, to provide the Central Directory feature that allows multiple "paths" to resolve to a single item such as a Client-ID, and to do this efficiently.

For more details, see HLD-x-1400 "Central Directory Service High Level Design"

11 Short message posting

The Short Message Posting (SMP) service accepts messages for delivery to client processes. These messages are limited in scope, in format and in function. The format of the message is limited to simple text messages with an associated administrative message for use by the client program. The message function is limited to a one way operation, no client replies are allowed, and to actions defined by the administrative message class.

The administrative message class instructs the client process to display a dialog box with a predefined set of choices. For example, a message can be sent to ask the user if the client program should shut down. The set of administrative messages is modifiable and extensible at run time. When new administrative message classes must are defined a new client plugin for the associated dialog box must be provided as well.

The Short Message Posting service is a simple rules interpretation engine. Rules are submitted in script form and result in a list of Client-IDs that represent message targets for a given message id. Each Client-ID item is associated with a time value indicating the time at which the message should be sent. Messages are stored in a database and referenced by message id.

The SMP service maintains a database of message target-time pairs for each posted message. The entries are ordered by increasing time delay so that a linear scan of a list will find the "next" target first. A separate list is maintained for the purpose of scheduling scans of the posted message lists. This list contains pairs of message ids and time values, message triggers, where the time value represents the next time at which the message is to be delivered to a client.

In addition to exact time values, priority encoding time values are supported. These employ a simple priority number scheme where a higher number indicates a higher priority. For message triggers with this time value type, the messages are sent as soon as all higher priority messages have been sent. A separate list of priority encoded message triggers is maintained for each message id so that their scanning will be orthogonal to exact time value encoding. Message ids can have both priority and exact time triggers.

A configuration option for the service controls the priority boundary. This boundary determines the priority relationship between priority valued message triggers and those with exact time values. If the time specified and in an exact time valued message trigger has arrived, and there is a message with a priority encoded message trigger pending, this value is used to determine which is sent first. If the priority trigger's priority value is greater than the priority boundary, then it is sent in preference to the exact timed message. If it is lower than the priority boundary, then the exact timed message is preferentially sent.

The SMP service maintains separate thread pools for sending messages and for processing rules. This means that urgent messages can be scheduled no matter how busy the service is with existing message traffic.

The rules submitted to the rules engine are able to take advantage of the features of the Central Directory to schedule messages. For example, all the clients on a particular branch of the network can be targeted exclusively. Alternately, this topology information might be used to schedule messages in waves, all the clients on one branch first, then the next, etc.

The SMP message scheduler implements one default rule for use when there are messages of equivalent priority or equivalent time waiting. In these cases the topology mapping features of the Central Directory are used to balance the load on the network. In other words, if ten message triggers are ready to fire, some for clients connected to "subnet5" and some for clients on "subnet3", then the trigger list will be reordered so that the triggers are interleaved. A message will be sent to a client on subnet5, then to one on subnet3, then to another one on subnet5, etc. This message trigger reordering is done when the message triggers are placed in the list so as to limit the delay during message processing.

12 Designed to be serviceable

A major challenge for any large-scale distribution architecture is serviceability, that is, the ability to check the health of the system and diagnose and correct problems. The highly dynamic nature of the ESB architecture and services makes this feature more critical, as the serviceability tools must track the changes in the system.

The ESB architecture has several features that address this challenge.

12.1 Service monitoring

The ESB Service Monitor subsystem consists of three components, the Service Configuration Management service, the Service Configuration Storage service and the Process Control service.

12.1.1 Service Configuration Management service

The Service Configuration Management (SCM) service provides an interface that is used to collect and manipulate the control information for ESB services both basic and extended. This data includes all the relatively static details of service configuration, such as how the services are grouped into servers, where these servers are running. This sort of data is stored in the Service Configuration Storage service.

The data is created by calls to the SCM, and in this way the implementation configuration of a service is defined. Configuration data falls into two general categories. The first is "start-time" data, which affects the target server or service only when it first starts executing. Any changes in this data will take effect only after a restart of the process. The second is "run-time" data, which has immediate effects. Whenever data in the run-time category changes, the SCM sends a message to the affected server(s) as a notification that the configuration data should be re-read.

More dynamic data is also collected directly from the services and reported by the SCM service. This includes details about how busy the server is, whether it is in danger of exhausting some resource such a disk space, performance statistics, etc.

The SCM provides reports on the information it collects in response to queries, but it can also execute scripts in response to events. For example, if a server stops responding to information queries, an event handler can run a script which changes the service configuration to eliminate the server and notifies an administrator.

12.1.2 Service Configuration Storage service

The Service Configuration Storage (SCS) supports optional replication, with servers distributed around the ESB. When this option is in use, the servers coordinate with each other to keep their data consistent so that the failure of one part of the infrastructure will not interfere with configuration management in other areas.

12.1.3 Process Control service

The Process Control service is a fully distributed service. Each server that hosts ESB server processes must have one (and only one) PC server running at all times. It uses methods appropriate to the platform to start

at boot time and restart if it fails for some reason. It collects information from the SCS, looking for servers that should be running on the host machine of that instance of the PC service. If the expected processes are not running, the PC uses the configuration data to start them, and it does this continuously until the configuration data indicates that the target server should not be running. Servers may bet specified for startup in sets, or in dependency trees.

This is a very simple, extensively tested process, so it is highly reliable. It provides the bootstrap capability for the ESB.

For more information, see HLD-x-1500 "Service Monitor Subsystem High Level Design"

12.2 Service tracing support

ESB services support a default level of tracing that records significant events, as defined by the logic of each service. This tracing is stored in memory in an efficient fashion, and is not normally sent to any output location. The trace control mechanism, which is part of the Service Configuration Manager and is in the category of run-time configuration, allows an administrative user to cause the trace data in memory to be sent to a variety of locations, including files and communications channels.

The trace control method supports the concept of trace classes, where messages are assigned to classes by the service developer. The pre-defined classes include warning and error classes, along with less urgent messages such as state changes, program events, etc. The trace class system is extensible, allowing the addition of either server wide or service specific classes.

Using these features, it is possible to control the tracing in any or all ESB services so that messages in classes that indicate trouble can be routed to one or more central monitoring programs, and other types of messages can be routed to a file, or not output at all.

One of the defined trace classes collects statistics about the interaction between the service and client programs. Each time a service request is received the tracing code collects address data about the client along with the time at which the request arrived. When the request has been serviced and the reply is sent to the client, the time of the reply is used with the other data to calculate the response time. In addition to creating normal trace data, this data is stored in a data structure that is designed to represent such events over time. The data is stored in an efficient way using little memory, but with enough structure maintained to allow useful statistical analysis.

There are two normal uses for this statistical performance data. One is to gage the health of the server, with regard to its ability to sustain throughput and response time levels. Another is to isolate pathological client behavior that is causing server overload.

12.3 Service probing

ESB services support a diagnostic probe function that is implemented as a server function. An administrative client program can send a probe request to a service or a server. The data returned by default is all the recorded trace information from a special "execution" class, which details the state of all the threads of execution. This information shows which threads are waiting for I/O, processing requests, and in other interesting states. This makes it possible to determine whether a service is executing normally or is in a pathological state.

The probe request can be sent with optional arguments that collect data from specific trace classes, or to collect the statistical performance data. By using a service specific extended trace class along with this probe feature, it is possible to implement any sort of environment or resource monitoring that the service requires.

For more information, see HLD-x-1600 "Serviceability Features High Level Design"